Capítulo 1

Maple

Este capítulo establece los conocimientos que necesita para trabajar en Maple 18.

1.1. Fundamentos

1.1.1. Terminología

Maple es un programa desarrollado desde 1980 por el grupo de Cálculo Simbólico de la Universidad de Waterloo (Ontario, Canadá). Su nombre proviene de las palabras *MAthematical PLEasure*.

1.1.2. Cualidades

- Es idóneo para realizar documentos técnicos
- hojas de trabajo interactivas basadas en cálculos matemáticos en las que puede cambiar un dato o una ecuación y actualizar todas las soluciones inmediatamente
- documentos estructurados
- permite estilos e hipervínculos,
- permite exportar a otros formatos como HTML, RTF, LaTeX y XML.
- permite realizar cálculos simbólicos complejos
- es un lenguaje de programación
- uso de unidades especializado

1.2. Entorno de desarrollo

1.2.1. Instalación del sistema

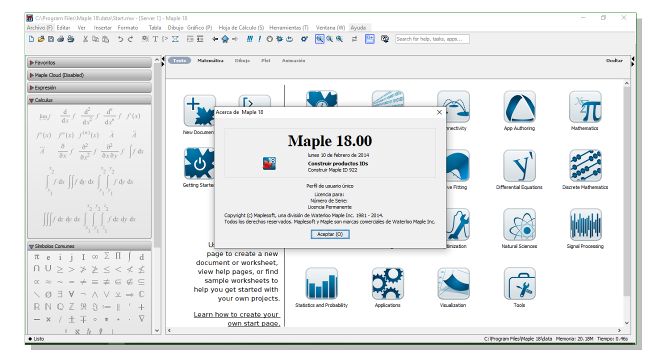
Engine `setup.exe`

Para instalar Maple 18.00 ejecute el asistente de instalación (setup.exe→P.179).

Engine `maplew.exe`

Este ejecutable es Maple 18 instalado. La Figura 1.1 muestra interfaz de usuario, identificación de la versión con la que trabaja este manual.

Figura 1.1. Interfaz de usuario Maple 18.00



Copyright Maplesoft, Waterloo Maple Inc. 1981–2014

1.2.2. Documentación

Engine `maplelaunchhelp.exe`

The Maple Help System [1] Figura 1.2; es la documentación oficial del lenguaje Maple (Menu UI: **Ayuda**>**Ayuda de Maple**, o Ctrl+F1).

Referencia normativa (manual de usuario) [2]; y, extendida (base) [3], [4], [5], [6].

1.3. Sistema Maple

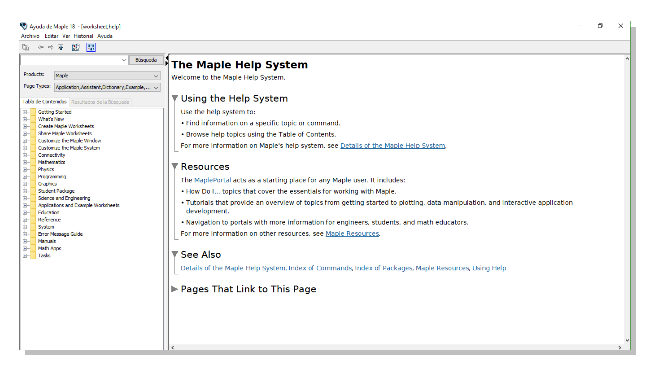
1.3.1. Mime type

Archivos de Maple:

File extension `*.mw`

Maple Worksheet.

Figura 1.2. Maple 18 Language Documentation



File extension *.mpl

Maple language files (contiene instruciones de programación lenguaje Maple).

1.3.2. Modos de trabajo

Para iniciar el trabajo (Menu UI: **Archivo**>**Nuevo**>) existen dos modos de trabajo [2, p. 3] *Modo hoja de trabajo* Figura 1.3 y *Modo documento* Figura 1.4.

Figura 1.3. Modo hoja de trabajo



Figura 1.4. Modo documento



1.3.3. Región

Lo que se escribe por teclado se interpreta según la región (o modo) [2, p. 2] en el que se encuentre el cursor, según la Figura 1.5:

A: Paletas

B: Sección

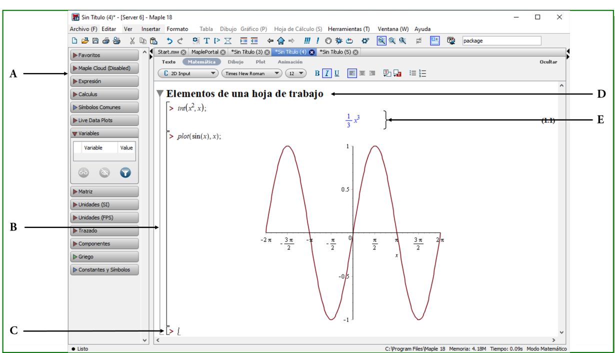
C: Prompt (Matemático)

D: Título de la sección

E: Grupo de ejecución

Puede dividir en secciones y subsecciónes desde el menú `Insert/Section`.

Figura 1.5. Regiones de edición Maple



Fuente: Aprenda Maple 9.5 [3]

1.3.4. Teclado

Pase a edición de texto normal, mediante Ctrl+T. Las regiones de texto son todas aquellas entradas que no requiren procesamiento como, anotaciones, descripciones, titulos ó hiperenlaces.

Pase a edición de una expresión matemática (o región de entrada matemática) mediante Ctrl+R. Las regiones matemáticas (Prompt) son para ingresar operaciones que requiere procesamiento matematico (la cual está activada por defecto).

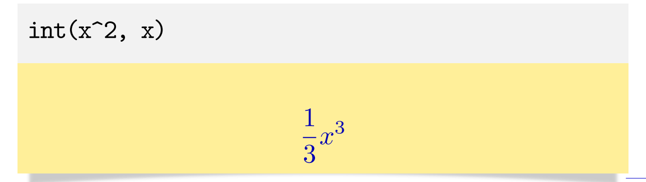
1.4. Matemática

1.4.1. Expresión matemática

Una expresión matemática puede ser escrita en modo comando (1-D) o en modo natural (2-D).

Modo texto (1-D)

Operación en 1D o entradas en formato comando (*Maple Notation*):



```
int(x^2, x)
```

$$\frac{1}{3}x^3$$

Capítulo 2

Operadores

2.1. Asignación

x := y (infix operator)

Asignación del valor y a la variable local x.

```
x := 45
```

$$45$$

2.2. Funcional

x -> y (infix operator)

Operador funcional [2, p. 75]. Donde x son las secuencias, nombres de variables, en tanto y es el resultado de aplicar el procedimiento.

```
f := x -> 3*x + 5;
f(2)
```

$$x \to 3x + 5$$

$$11$$

Internamente la operación funcional es equivaltente usando `proc`.

```
f := proc (x) options operator, arrow; 3*x+5 end proc;
f(2)
```

$$x \to 3x + 5$$

$$11$$

2.3. Tipo

x :: y (infix operator)

Declaración de tipo y del parámetro de un operador funcional de variable x. En adición, también puede usarse en una *afirmación* (—assertion) de tipo, cuando se habilitan mediante `kernelopts(assertlevel = 2)`.

```
f := ( x::integer ) -> 3*x + 5;
f(2)
```

$$x :: integer \to 3x + 5$$

$$11$$

```
f := ( x::integer ) -> 3*x + 5:
f(2.3)
```

```
Error, invalid input: f expects its 1st argument, x, to be of type integer, but received 2.3
```

```
kernelopts(assertlevel = 2)
x::integer := 3.4
```

```
Error, assertion failed in assignment, expected integer, got 3.4
```

2.4. Aritméticos

x + y (infix operator)

Suma x a y.

`2 + 3`= 5

x - y (infix operator)

Resta x a y.

`2 - 3`= -1

x * y (infix operator)

Multiplica x por y.

`2 * 3`= 6

x / y (infix operator)

División de x por y.

`7 / 3`= $\frac{7}{3}$

x ^ y (infix operator)

Potenciación x^y.

Capítulo 3

Funciones

3.1. Identificadores

3.1.1. Reservados

Identificadores del sistema estan protegidas (`protect`→P.13) como el caso de `sin`→P.17.

```
sin:=258;
```

```
Error, attempting to assign to `sin` which is protected.  Try declaring `local sin`; see ?protect
    for details.
```

3.1.2. Protección

`protect`(⟨*name*⟩,⟨*name2*⟩,. . .)

Protege o desprotege el identificador ⟨*name*⟩ [2, p. 75]. Una variable protegida (o reservada) no puede ser redefinida.

```
data:=258;
protect('data');
data:=45;
```

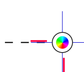
258

```
Error, attempting to assign to `data` which is protected.  Try declaring `local data`; see ?
    protect for details.
```

`unprotect`(⟨*name*⟩,⟨*name2*⟩,. . .)

Desprotege la variable ⟨*name*⟩.

```
unprotect('sin');
sin:=45;
2*sin+4;
```



$$45$$

$$94$$

3.1.3. Persistencia (unassign and restart)

`unassign`(⟨*name*⟩,⟨*name2*⟩,...)

Borra el valor de la variable local ⟨*name*⟩.

```
x:=43:
x^2+2x+3;
unassign('x');
x^2+2x+3;
```

$$1938$$

$$x^2 + 2x + 3$$

`restart`

Restablece la memoria interna [2, p. 68, 76] del sistema a su configuración default. Esto borra todas las asignaciones locales definidas por el usuario, incluyendo las `protect`→P.13, `unprotect`→P.13 y los identificadores externos (importados con `with`→P.16).

```
x:=43:
x^2+2x+3;
restart;
x^2+2x+3;
```

$$1938$$

$$x^2 + 2x + 3$$

3.1.4. Evaluación

`eval`(⟨f⟩, ⟨x⟩=⟨a⟩)

Evaluación de las expresiones. $\langle f\rangle\Big|_{\langle x\rangle=\langle a\rangle}$ Este método viene sobrecargado

- `eval[recurse]`(⟨*e, eqns*⟩)
- `eval`(⟨*f, n*⟩)

```
% eval(e, x=a)
% eval(e, eqns)
% eval[recurse](e, eqns)
% eval(e)
% eval(x, n)
```

Print Preview (Beta Test) ~ Presentación Preliminar